The 47th IEEE International Midwest
Symposium on Circuits and Systems

An Efficient Algorithm/Architecture Codesign for Image Encoders

Jinku Choi
Dept. of Computer Engineering, Korea Polytechnic University

Nozomu Togawa
Dept. of Information and Media Sciences, The University of Kitakyushu

Takeshi Ikenaga and Satoshi Goto
Dept. of Graduate School of information, Product and Systems Waseda University

Masao Yanagisawa and Tatsuo Ohtsuki
Dept. of Electronic, Information and Communication Engineering Waseda University

Abstract: We describe the optimization of a complex video encoder systems based on target architecture. We implemented the MPEG-4 encoder using hardware/software codesign approach, mapped together based on a target architecture. We proposed a target architecture template and an optimization methodology. In our design flow, we searched for a bottleneck module constraining the system. After investigating the computational complexity, quality, and the simplicity of algorithms, we chose the best algorithm for hardware implementation, and then mapped the selected algorithm onto the hardware with different architecture, what does the best architecture for the algorithm and which is the best architecture of components. We chose one of the architectures meet the constraints and also made tradeoffs among speed, chip area, and memory bandwidth for different architecture. The proposed system architecture was used to reduce the design decisions and iterations, provided flexible and scalable systems. The evaluations resulted in effective optimization of the motion estimation module and better tradeoffs that optimized the overall system.

I. Introduction

New design methods are needed to deal with such complex systems and meet demands. Hardware and software codesign is emerging as a promising approach to deal with these challenges. Recently, hardware/software codesign has moved to a higher level of abstraction through the platform based and the reuse of IP in embedded systems. During design processing, the designer makes design decisions, which must be based on quantitative simulations and evaluations of the results.

However, these design decisions are often inefficient, sub-optimal, or occur too late in the design process. Decisions involving the algorithm and architecture or hardware and software for performing a function affect system performance, chip area, and power consumption.

Often, various algorithms and architectures are available for the same a function or an application, but their performance and complexity differ markedly.

In this paper, we describe efficient feedback between algorithm and architecture design, which involves tradeoffs that can be effectively analyzed at the pre-implementation level before decisions must be made. We investigated the computational complexity and quality of motion estimation algorithms and the simplicity of hardware implementation.

Then, we chose and implemented the best algorithm. To obtain hardware optimization, we used a parallel PE array.

We were able to make tradeoffs and optimize the design using information based on evaluations of different constraints placed on the encoder and different applications.

II. Related Works

In the Y-chart scheme [1], the designer first characterizes a set of applications and chooses an architecture to run the set. Then, each application is mapped onto the architecture and the performance of each application/architecture mapping is evaluated. Depending on the resulting performance, the designer may decide to use that architecture, and then restructure the application, or modify the mapping of the application to achieve better performance.

The CoWare [2] provides a methodology along with an architecture template. The signal-processing kernels are specified as functions. The interfaces between program and kernels are automatically generated. The architecture is abstracted ad an interconnection of processor component units with point-to-point communication channels. This has a point-to-point generated physical interconnection network, but limited ports.

Here, we propose a hardware/software codesign methodology, based on algorithm and architecture codesign, using a system architecture template.

III. Hardware/Software Codesign Flow

Our design flow describes the design flow for an actual system, which exists as various algorithms and architectures in the video applications. The basics of our codesign flowchart are shown in Figure 1.

① The design methodology starts with the system specifications, constraints, and requirements.

② Then, the intended functions of the system are decomposed, to give a model of the application from the design specifications. A functional description is structured in the language. In order to obtain an estimate of the computational complexity, the language code is ported on the system of architecture. We searched a bottleneck module among functional modules.

③ In this step, process of selecting the best algorithm. We optimized algorithm of bottleneck module in software domain. We select the best an algorithm among candidate algorithms based on simulations and evaluations to optimization the bottleneck module. The selection process assumes the existence of multiple algorithms for implementing the same function.

④ A given algorithm can be implemented using many architecture solutions and various approaches to hardware design. The selected algorithm is then mapped on the candidate hardware architecture. During this time all the necessary hardware-hardware, hardware-software, and software-software partitioning is carried out manually, to specify whether a function block is to be implemented in hardware. The selected hardware functional blocks are clustered into dedicated hardware modules. The functional processes are mapped onto architectural resources on the target architecture.

⑤ In this step, the selected algorithm mapping onto hardware architecture and optimize hardware architecture. The candidate architecture has the potential to realize its

0-7803-8346-X/04/$20.00

function, including the choice of IP blocks or virtual modules in hardware domains. A few more general design decisions can be taken at an early design process stage (②-④), while more detailed decisions are made at a later stage. We optimize functional modules, the bus and system architecture are fixed using architecture template. The refinement phase involves performance and other studies to optimize the architecture to realize the function, and the architectural structure or system functionality are modified until the design becomes feasible.

⑥ In this step, the hardware and software are implemented for each module in the system, and the interfaces between hardware-hardware, hardware-software, and software-software are implemented.

⑦ Finally, hardware-software coverification for both the hardware and software portions of the system is carried out on the architecture platform.

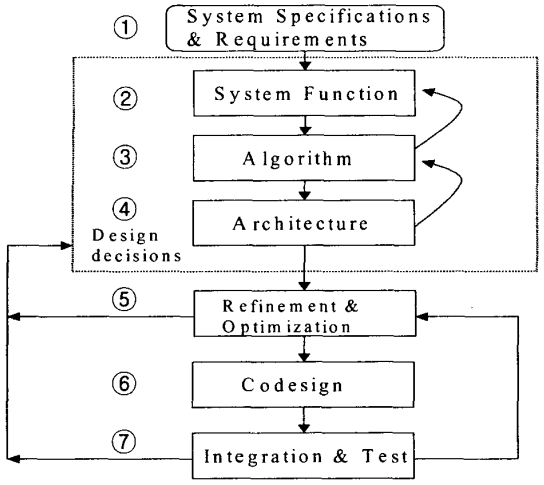

Figure 1 Hardware and Software Codesign Flowchart

IV. ARCHITECTURE PLATFORM

Our proposed architecture template is depicted in Figure 2.

The proposed architecture is based on a programmable RISC processor, and the AMBA (advanced microcontroller bus architecture) bus[3], shared main memory, and hardware modules with local memory. The programmable processor and hardware modules are connected to the AMBA local bus via the main memory. CTRL is the interface bus between the hardware modules and the AMBA bus. The dedicated hardware modules are the functional modules labeled HW1, HW2, and HW3.

The local memory consists of the local data, and the buffer for the dedicated hardware modules. For high performance data processing, the modules need to have a high bandwidth to memories. The modules have to access memory via a system bus, AMBA bus, in this case the bus becomes a bottleneck. Therefore, we choose to local memories over the hardware modules. The software tasks are executed on the programmable processor and the hardware tasks are executed on the dedicated hardware modules. Our architecture template is very simple and more practice as actually design goal.

The target architecture should allow a variety of processing elements, RISC processor families and this architecture can be to rapidly extended, customized for a range image encoder applications, and speedup design process.

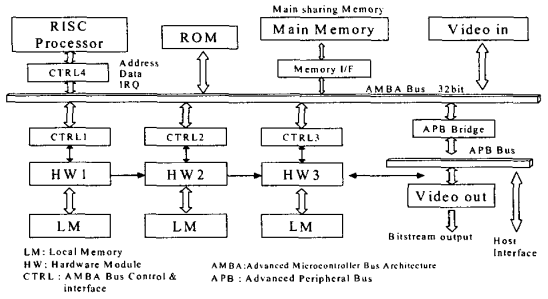

Figure 2 System Target Architecture

V. CASE STUDY

A. *Design Specifications of MPEG-4 Encoder*

In this paper, we assumed our target design specifications, to implement the MPEG-4 encoder, listed in Table 1.

Table 1 The MPEG-4 Encoder Design Specifications

- Simple Profile Levels 0,1,2 and 3.
- Resolution up to CIF encoding at 30 fps.
- 4:2:0 YCrCb input.
- DCT and IDCT compatible with IEEE Std 1180-1990.
- Quantization and Inverse Quantization (H.263).
- Zigzag-SCAN, Run length coding (RLC).VLC.
- ME (Motion estimation).
  ; Range of motion vectors 16 pixels.
  SAD matching criteria.
  1 MV / macro block.

B. *System Functions*

In this design step, one or more operations in the MPEG coding sequence (motion estimation, DCT, quantization, VLC coding, etc.) are assigned to a different processing unit.

The each processing unit can be optimized for the given task for a system optimization.

MPEG-4 encoding includes several computation intensive algorithms, such as DCT and motion estimation.

C. *Software Algorithm Implementations*

We selected a RISC processor type, ARM-9, in the architecture template. Using the C language, we designed and compiled it based on ADS (ARM developer suite) Ver.1.2 [4]. Then, we examined the assembler code generated. For the implementation software, we calculated the total number of instruction cycles per macroblock and the size of ROM based on the ARM-9 RISC processor in the system architecture template. The processor execution time of a macroblock is the total number of instruction cycles x clock cycle time. We estimated ROM area from ROM data and instruction size. The results are summarized in Table 2.

Table 2 The Implementation of MPEG-4 Encoder.

Modules	Hardware implementation		Software implementation	
	Chip area (mm^2)	Execution time/MB (*nsec*)	ROM area (mm^2)	Execution time/MB (*nsec*)
RGB2YCrCb	0.312499	2,189	0.401	20,680
DCT/IDCT	0.423225	4,382	0.675	44,080

MEMC	0.817256	15,211	1.560	164,320
Q(DC/AC)/IQ	0.467216	1,397	0.401	28,160
VLC	0.746935	1,658	0.174	16,840

D. Comparison and Evaluation of Algorithms

We investigated computational complexity for motion estimation algorithms. We analyzed complexity based on the maximum number of checkpoints for the candidated algorithms. It can be seen that the number of checkpoints is dependent on d (the maximum displacement of a motion vector) and block size (M x N). Although the number of checkpoints depended on the characteristics of images, the BBGDS algorithm had the fewest check-points of the algorithms evaluated.

Our proposed algorithm [5] has 27 maximum checking-points at d=7 and d=15. The BBGDS, NTSS, and FSS algorithms in the worst case require 24, 33, and 27 checkpoints at d=7, respectively, while the TSS requires 25.

We evaluated the motion estimation algorithms, implemented them, and analyzed image quality and CPU speed. Table 3 shows the average PSNR (peak-signal-noise rate) and CPU speed.

There are some differences between the number of checkpoints and CPU speed, because the BBGDS, NTSS, and FSS algorithms reuse memory data, reducing the number of checkpoints for center-biased motion sequences, while TSS must access memory data in all search areas without reusing data. Therefore, TSS is slower than the BBGDS, NTSS, and FSS algorithms for all image sequences, especially, for very center-biased motion sequences, depend on image characteristics.

Quality depends on the characteristics of the image sequences type and applications. Before we knew that the bottleneck module is ME. We focused on motion estimation algorithm to optimization the encoder. In this step, we chose an algorithm among algorithms based on complexity, quality, and simplicity of implementation hardware.

We selected our proposed algorithm [5] for hardware implementation to optimize the encoder.

Table 3 The Software Simulation Results of the five Algorithms in terms of the Average PSNR and the Speed.

Images	Algorithm	Average PSNR(dB)	Speed (sec)
Football (352x240) 80 frames	FS	26.49	313.71
	TSS	26.01	13.96
	BBGDS	26.13	6.59
	FSS	26.12	12.72
	NTSS	26.10	9.82
	Our	26.16	7.26
HALL (176x144) 80 frames	FS	35.41	59.65
	TSS	35.36	3.04
	BBGDS	35.37	1.47
	FSS	35.37	2.84
	NTSS	35.34	2.19
	Our	35.37	1.58

E. Hardware Implementations

We designed the hardware using the VHDL, Synopsys Design Compiler Tools, and 0.35μm Hitachi CMOS technology, and obtained the maximum data delay and area. The hardware implementation results are summarized in Table 2.

Table 4 The number of Cycles, Memory Bandwidth Required, and Chip Area for the PE Architecture. (The clock rate is constrained to 20 nsec).

PE array	No. Cycles /MB	Memory bandwidth (Byte/s)	Chip area(mm^2)
9x1	768	7680	0.60481
3x1	2304	3072	0.251782
9x2	384	7296	1.283611

F. Architecture Implementations

We investigate what does the best architecture for an algorithm, which is the best architecture of a component, and then we do tradeoffs. For a given algorithm, there are many architecture solutions for motion estimation implementation and there are various approaches to hardware design. On the way from an algorithm to an efficient architecture, complex decisions must be made. Often it is not efficient, which architecture decision is the best.

From investigating our proposed algorithm, we knew that it consists of three nested loops. We can parallel loop 2 for each of the nine candidate positions by using nine PEs with a single PE calculating the SAD for one of the nine search positions and sequentially executing operations in each search step (loop 1) and pixel calculation (loop 3).

The architecture shows the basic PE array structure and the input data flow. The current data are sequentially broadcast to all nine PEs simultaneously from a single data bus and are identical for all nine PEs. Since the reference data differ among PEs, the reference data are broadcast from each memory bank to a PE through separate data buses. The next step address provides the base address for the next step when the PE is calculated at each of the nine checkpoints, and then the correct address can be selected immediately once the minimum SAD is determined.

The architecture requires up to 256 cycles at each search step to calculate the SAD. With this architecture, each macroblock can be processed in 768 cycles.

The memory bandwidth is 7680 byte/s. We need to find a reasonable tradeoff and to optimize the system in terms of parallelism, memory bandwidth, control and computational cost, array size, cycles required, etc.

We can also configure the system using differences in the architecture of the PE arrays and the number of PEs. These results are shown in Table 4.

G. Optimization

From the perspective of hardware implementation, we must consider the worst-case situation. When the algorithm is implemented using hardware, there are different design results.

Our proposed algorithm is simpler and has good data flow regularity, and a low control overhead compared to the other algorithms. To obtain the high processing performance required for executing motion estimation algorithms, two principles are fundamental to any architecture: pipelining and

parallelism. The parallelism approach is especially well suited for image processing, which relies on block-based algorithms.

Parallelism is possible at the pixel, algorithm, and macroblock levels, and pipelining at the pixel and algorithm levels. Figure 3 plots the number of cycles vs. chip area for each of PE architecture, which consists of 3, 9, 18, and 27 PEs.

For example, given the constraints indicated by the dotted line rectangle for the number of cycles and chip area, we can select an N=9 architecture as the best tradeoff.

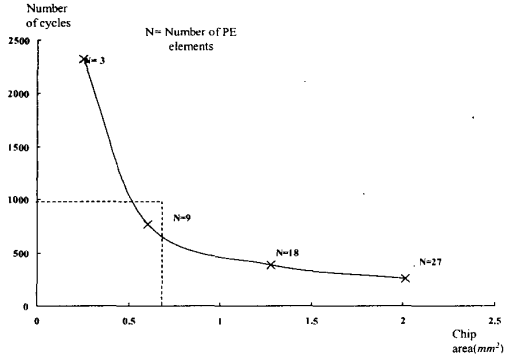

Figure 3 The number of Cycles, Required Memory Bandwidth, and Chip Area According to the Number of PEs (The clock rate is constrained to 20nsec)

H. Implementation Results

In the MPEG-4 video encoder, there are two groups of algorithms with very similar basic properties can be determined; one group is macroblock-processing oriented algorithms, like ME, DCT/IDCT, The other group is stream processing oriented algorithms, like VLC. It is significant differences in data width and parallelization potential between the groups.

We summarize our implementation results in Table 5. The architecture of the encoder is shown in Figure 4. We mapped the system architecture template in Figure 2 onto Figure 4. HW1 is mapped to the ME/MC module; HW2 and HW3 are mapped to the DCT/IDCT and Q/IQ, respectively. The VLC is mapped in software more efficient than mapped in hardware module. Table 2, VLC is the biggest difference value between hardware and software results. Software implementation is has an advantage over hardware and VLC processes based on stream data.

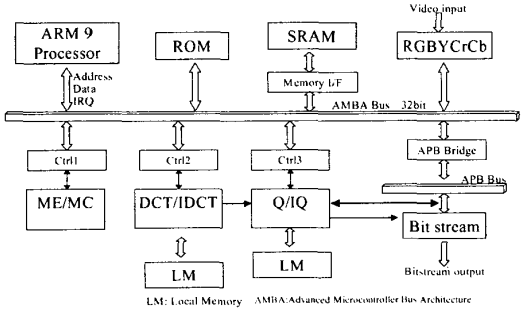

Figure 4 Mapping the MPEG-4 Encoder onto Target Architecture

Table 5 The Implementation Results.

Modules	HW or SW	Chip area (mm2)	Execution time/MB (nsec)
RGB2YCrCb	HW	0.312499	2,189
DCT/IDCT	HW	0.423225	4,382
ME/MC	HW	0.817256	15,211
Q/IQ	HW	0.467216	1,397
VLC	SW	0.174	16,840
CTRL	HW	0.268372	146
ARM9DTMI	SW	4.15	

VI. CONCLUSIONS

We proposed an implementation method for an MPEG-4 encoder based on codesign of the system algorithm and architecture using the system architecture template. In our codesign flow, we analyzed the complexity and performance of MPEG-4 encoders. Then, we searched for the bottleneck module, and therefore focused on improving this module.

We also investigated several motion estimation algorithms, and then chose the best algorithm for hardware implementation. We mapped the selected algorithm onto the hardware using different architectures, and chose one of the motion estimation architectures, making tradeoffs between speed and chip area for the different applications.

We presented relations the performance, chip area, and required memory bandwidth according to the number of PEs. The tradeoffs between algorithm and architecture must be optimized to deliver a design with performance and area constraints that are satisfactory.

Our proposed system architecture can be easily expanded to other image applications and multi-standard video encoder systems, and provides a flexible, scalable system using a standard system bus.

Our approach can be reduced the design decisions and shorten the design time.

In future work, we will evaluate power consumption for each module and performance, which consider to scheduling of system.

ACKNOWLEDGMENT

REFERENCES

[1] P. Lieverse, P. V. d. Wolf, E. Deprettere, and K. Vissers. "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems," *IEEE Workshop on Signal Processing Systems*, Taipei, Oct. 1999.

[2] V. Rompaey, R.D. Verkest, I.Bolsens, and H. de Man, "CoWare - A Design Environment for Heterogeneous Hardware/ Software Systems," *Design Automation for Embedded systems Conference*. pp.357-386,1996.

[3] AMBA 2.0 Specification, ARM,http://www.arm.com/armtech/

[4] ADS 1.2 Development Guide, ARM, http://www.arm.com/arm/

[5] Jinku CHOI, Nozomu TOGAWA, Masao YANAGISAWA, Tatsuo OHTSUKI., "An Algorithm and a Flexible Architecture for Fast Block-Matching Motion Estimation," *IEICE Trans.on Fundamentals of Electronics Communications and Computer Sciences*, Vol.E85-A. No.12,pp.2603-2611, Dec.2002.